Mini-CCNx: Fast Prototyping for Named Data Networking

Carlos M. S. Cabral, Christian Esteve Rothenberg, Maurício Ferreira Magalhães
Faculty of Electrical and Computer Engineering (FEEC)
University of Campinas (UNICAMP), São Paulo, Brazil
{cabral,chesteve,mauricio}@dca.fee.unicamp.br

ABSTRACT

Experimental research in Information-Centric Networking (ICN) is crucial to the evaluation of new architectural proposals that bring named pieces of content as the main element of networks. This paper presents a new fast prototyping tool for the NDN (Named Data Networking) model, Mini-CCNx, that aims at filling an existing gap in generally available experimental platforms. Using container-based emulation and resource isolation techniques, Mini-CCNx appears as a flexible, scalable, high-fidelity, and low-cost tool that enables rich experimentation with hundreds of emulated NDN nodes in a commodity laptop.

Categories and Subject Descriptors

C.2.1 [**Network Architecture and Design**]: Network communications

Keywords

ICN, NDN, CCN, emulation, prototyping, routing

1. INTRODUCTION AND GOALS

Experimentally driven research in Information-Centric Networking (ICN) is crucial to the evaluation of open research issues in this area, such as routing protocols, forwarding strategies, caching techniques, content-centric application development and so on. During our research journey on new forwarding strategies and probabilistic state reduction techniques for NDN core nodes, we faced an existing gap in feature-rich, generally available experimental tools (cf. Table 1). We could not find a low-cost, scalable, high-fidelity, and sufficiently flexible experimental platform to carry in-depth evaluation of diverse and customizable NDN scenarios. The lack of a tool combining all desired factors and the lessons learned with OpenFlow prototyping motivated the developments of Mini-CCNx.

ICN'13, August 12, 2013, Hong Kong, China.
ACM 978-1-4503-2179-2/13/08.

Table 1: Summary of NDN experimentation tools.

Ex:	**Simulators** *ndnSIM/ccnSim*	**Testbeds** *Testbed NDN*	**Emulators** *Mini-CCNx*
Flexibility	High	Low	High
Scalability	High	Low	High
Cost	Low	High	Low
Realism	No	Yes	Yes
Ease of Config.	Medium	Low	High
Link config.	Yes	With restrictions	Yes

2. MINI-CCNx ARCHITECTURE

Mini-CCNx is a fork[1] of Mininet-HiFi [4] –originally proposed for OpenFlow networks– augmented with several classes and mechanisms to build NDN environments based on the project's official code base [6]. Mini-CCNx uses container-based emulation (CBE), a lightweight OS-level virtualization technique. Each container allows groups of processes to have independent views of system resources, such as process IDs, file systems and network interfaces while still using the same kernel. Each container is a NDN node, with its own network namespace, virtual network interfaces, NDN-specific data structures (PIT, FIB, and CS) implemented by the `ccnd` daemon, and repositories as implemented by the `ccnr` daemon. These nodes are connected to each other using virtual Ethernet links in the kernel space. One major challenge Mini-CCNx faces is related to performance isolation and the effectiveness of isolation techniques to limit the resources available for each node and link to guarantee high-fidelity results. To this end, Linux `cgroups` [3] are used to limit CPU bandwidth for each node. Mini-CCNx also adds

[1]Available at: https://github.com/carlosmscabral/mn-ccnx

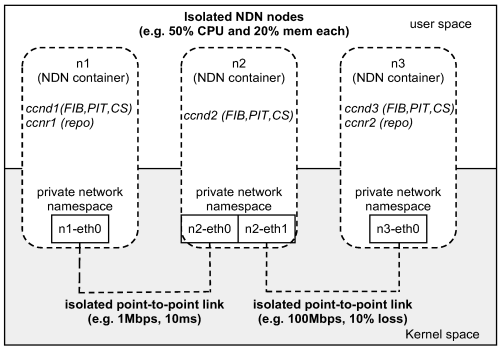
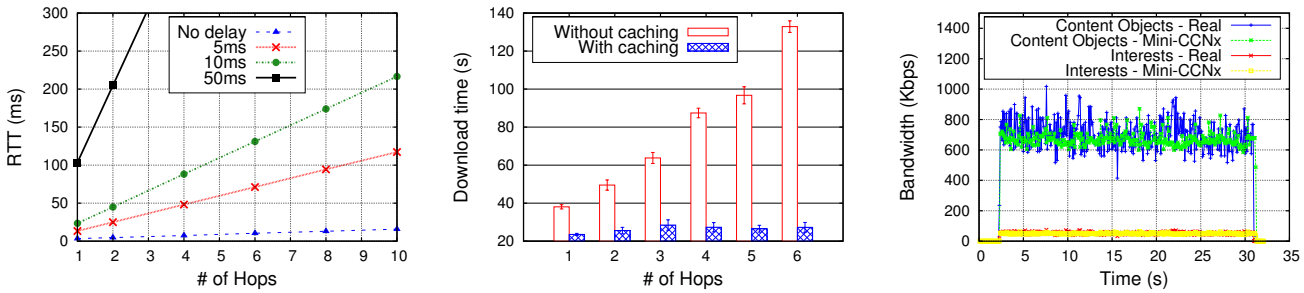


Figure 1: Three NDN nodes connected using Mini-CCNx containers



(a) RTT vs # hops for varying link delays (b) Average download time vs # hops (c) BW comparison with real testbed

Figure 2: Coherence and fidelity analysis

memory limits, a relevant subject for NDN when it comes to content caching and FIB/PIT scalability. Finally, using `tc`, it is possible to configure several link properties such as bandwidth, delay, and packet loss. Figure 1 illustrates the CBE and isolation features used by Mini-CCNx.

3. PERFORMANCE AND FIDELITY

We evaluate Mini-CCNx's fidelity and performance using a low-cost hardware, largely available to most researchers and students.[2] The `CCNx` version used in all cases is 0.7.0. We analyse the following performance dimensions: (*i*) scalability, (*ii*) coherence, (*iii*) fidelity, and (*iv*) isolation. A more detailed analysis can be found in the Mini-CCNx technical report [1].

1) Scalability. Mini-CCNx performs really well, being able to instantiate hundreds of nodes in different topology arrangements (linear, mesh) in a relatively short time (e.g. about 5 minutes for 1536 NDN nodes in a linear topology).

2) Coherence. We focus now on how coherent are the results with regards to experiment parameters such as link delay, bandwidth, and number of hops. For this analysis, we investigate two simple NDN scenarios. In the first one, we measure the RTT using `ccnping` [6] for varying hop numbers and link delay values (Fig. 2(a)). As expected, the RTT increases linearly with the number of hops. For different link delay values, the RTT behavior is consistent. For example, under 10 ms link delay and for 2 hops, the *ping* takes 20 ms upstream, plus 20 ms downstream, plus some processing time per node. In the second scenario, the average download time of a 100MB file was measured for different hop distances. Two sub-cases were analysed: (1) *no-caching*, where Interest messages go all the path until the producer, and (2) *caching*, where a first download of the file populates node caches along the path, and then a second request may retrieve pieces of content from the caching nodes (default caching configuration of CCNx was used). As shown in Fig. 2(b) (with 95% CI), download times increase for larger distances between client and producer. We can also observe the expected effects of caching, where the download time benefits from the amount of cached content in the nodes closer to the client.

3) Fidelity. As a first assessment on how Mini-CCNx is able to reproduce real experiments with high-fidelity, we use a simple topology with two real desktops and native `ccnx` installations directly connected via FastEthernet interfaces. Using the `ccntraffic` [2] generator, the first desktop constantly sends Interest messages asking for different contents while the second desktop answers with 1024-bytes Content Objects. The same scenario was reproduced using Mini-CCNx, with 100Mbps and a delay of 200 μs as link parameters. Figure 2(c) shows the Interest and Content Objects (Data) traffic for both cases. Note that the bandwidth consumption in Mini-CCNx follows very closely the real deployment behavior.

4) Isolation. A thorough isolation analysis can be found in [1], where we show that the correct resource allocation using *cgroups* greatly reduces the effects of the interference faced by the emulation system (e.g., influence of cross-traffic on PING RTT measurements).

[2]All the tests were done on a laptop with a typical configuration (Intel Core I5 2410M processor with 4GB RAM).

4. CONCLUSION

Mini-CCNx offers an experimenter-friendly emulation environment that includes many configuration options (e.g., topology, link parameters, FIB manipulation, CCNx apps) in addition to a GUI and available NDN software artefacts (e.g., NDNVideo, OSPFN). As detailed in [1], we have been able to reproduce (with great fidelity) several results from the NDN literature [5, 7]. We believe that developers and researchers may benefit from Mini-CCNx for fast prototyping purposes and to gain early insights on application and protocol behavior prior to costly and complex testbed deployments.

5. REFERENCES

[1] C. Cabral, C. Esteve Rothenberg, and M. Magalhaes. Mini-CCNx Tech Report TR-001-Mini-CCNx. `http://github.com/carlosmscabral/mn-ccnx/wiki/Publications`, Aug. 2013.

[2] CCNx Traffic. CCNx: traffic generation - Wiki. `http://wiki.arl.wustl.edu/onl/index.php/CCNx:_traffic_generation`.

[3] cgroups. Linux Control Groups. `https://www.kernel.org/doc/Documentation/cgroups/cgroups.txt`.

[4] N. Handigol, B. Heller, V. Jeyakumar, B. Lantz, and N. McKeown. Reproducible network experiments using container-based emulation. *CoNEXT '12*, page 253, 2012.

[5] V. Jacobson, D. K. Smetters, J. D. Thornton, M. F. Plass, N. H. Briggs, and R. L. Braynard. Networking named content. In *CoNEXT '09*, Dec. 2009.

[6] Named Data Networking. NDN Github Repository. https://github.com/named-data, April 2013.

[7] NDN Project. NDN Technical Reports. `http://www.named-data.net/techreports.html`.